Praktické základy SQL

V SQL
je fakticky jen jeden příkaz,
a tím je příkaz SELECT

SQL je jazyk původně určený pro
agenty FBI a sekretářky

Autor: Pavel Stěhule ©2013
pavel.stehule@gmail.com

Databáze

- Slouží pro uchování informací - aktuálních případně i historických
  - seznam telefonních čísel
  - ceny produktů
- Slouží pro získání informací z uložených dat
  - řazení - 10 nejprodávanějších produktů
  - agregace - získání minimální/průměrné/maximální ceny
- Každá databáze by měla měla mít svůj účel, své definované hranice

Relační databáze

- Základním kamenem je databázová relace
  - matice skládající se z řádků a sloupců
  - všechny řádky v matici mají stejný počet sloupců
  - co řádek, to entita - objekt v realitě
  - co sloupec, to atribut - určitá vlastnost reálného objektu - tím je určen i typ a rozsah hodnot (celé číslo, datum, text, ...)
  - **tabulka je relací** - která je uložena trvale na disku
  - výsledek příkazu SELECT je relace
  - vstupem pro příkaz SELECT jsou relace.

Databázová relace

- Každý řádek má stejný počet sloupců, což odpovídá jednomu záznamu s jeho atributy
- Hodnoty v jednom sloupci mají stejný význam a stejný datový typ - Pokud hodnotu neznám, mohu použít NULL.
- Pořadí řádků v relaci je nedůležité, náhodné, nedefinované.

Osobní číslo	Jméno	Příjmení	Narozen	Adresa
1	Pavel	Stěhule	15.7.1973	Skalice 12,..
2	Zdeněk	Stěhule	28.12.1978	Skalice 12,..
3	Michal	Prázdný	NULL	NULL

SQL relační databáze

- Jazykem pro operaci s daty je jazyk SQL, pro který existuje norma ANSI SQL
- definice - CREATE TABLE
- manipulace s daty - INSERT, UPDATE, DELETE
- dotazování - příkaz SELECT

Rozdělení SQL databází

- databáze pro OLTP
  - transakční zpracování 80/20 SELECT/UPDATE
  - více uživatelů může editovat souběžně stejná data
- databáze pro OLAP
  - analytické zpracování 99/1 SELECT/UPDATE
  - uživatelé pouze analyzují data

SQL přílazy

- CREATE TABLE foo(a int);
- INSERT INTO foo VALUES(10);
- SELECT * FROM foo;
- UPDATE foo SET a = 20;
- **SELECT * FROM foo WHERE a = 20;**
- DELETE FROM foo;
- *POZOR! pouze příkazy SELECT nebo úpravy pouze vlastních tabulek a dat!* ***Správné aplikace jsou zabezpečené proti nedovolené změně dat, ale je Vaše aplikace správně zabezpečena?***

Cvičení

- Vyzkoušejte si SQL příkazy v předchozím slajdu
- Vyzkoušejte si příkazy \d a \dt
- Vyzkoušejte si historii příkazů

Kdy použít relační DB?

- Pracujeme se strukturovanými daty
- Pracujeme s velkými objemy dat

Databáze nenahrazují tabulkové procesory, které slouží k modelování (analýzám WHATIF). Zrovna tak by tabulkové procesory neměly nahrazovat databáze - pro uložení stejného objemu dat mají významně větší požadavky na hw.

Normalizované schéma

- **Tabulky jsou relace** - jen jsou uložené na disku.
- Data na disk chceme ukládat efektivně, bez duplicit - chceme minimalizovat čtení a zápis, chceme maximálně zjednodušit aktualizaci.
- Tabulky, kde nedochází k duplicitám vyhovují tzv **normálním formám** - sada pravidel popisujících ideální databázi (v několika úrovních). Každá úroveň normálních forem vyžaduje splnění svého pravidla a všech pravidel nižších úrovní.
- Pro dobře navrženou databází s normalizovaným schématem se dobře píší SQL příkazy.

Normální formy

- 1. NF - každý atribut obsahuje pouze atomické hodnoty - jméno | příjmení.
- 2. NF - každý neklíčový atribut závisí na celém klíči - nedochází k opakování hodnot.
- 3. NF - atributy jsou vzájemně nezávislé.

V důsledku aplikace normálních forem jsou data rozdělena do více tabulek.

Téměř vždy musíme pro získání žádaného výsledku pracovat s více tabulkami.

Porušení 1.NF

- uložení jména a příjmení do jednoho atributu
- zneužívání atributu poznámka
- jakmile musíme složitě získávat zajímavé hodnoty z atributu, tak máme tento problém ..

Porušení 2. NF

- Pokud se nám v databázové relaci opakují neklíčové atributy tak máme problém s porušením 2. NF
  - při aktualizaci musíme upravovat více řádků

Kód produktu	Název	Kód dodavatele	Dodavatel	Telefón
1	Tmavý chléb	BNP	Benešovské pekárny	317777272
2	Bílé pečivo	BNP	Benešovské pekárny	317777272

Klíčem (jednoznačně identifikující záznam) jsou sloupce ("kód produktu", "kód dodavatele") - nicméně dodavatel a telefon závisí pouze na "kódu dodavatele" - nikoliv na "kódu produktu". Běžná chyba pro uživatele Excelu, kteří používají schéma v tzv 0.NF

Porušení 3. NF

- dopočítávané sloupce
  - při aktualizaci nesmíme zapomenout na dopočítávané sloupce

kusů	cena	cena celkem
10	20	200
10	30	300

Pokud je výpočet dopočítávané hodnoty složitější a může se měnit v čase, pak se 3NF porušuje - takové hodnoty ale málokdy zpětně upravujeme - např. výpočet mzdy, daní, ...

Pohledy

- Databázové relace určené primárně pro čtení - zobrazují pohledy (zajímavé kombinace dat) do persistentních relací (tabulek)
- Navržené tak, aby obsahovala data ve formátu vhodném pro práci uživatele
- Navržené tak, aby obsahovala data dostupná pro uživatele (omezení viditelnosti)
- Mohou porušovat normální formy

Referenční integrita

- Klíč - sloupec (sloupce) jednoznačně identifikující záznam.
- Primární klíč - jednoznačně identifikuje záznam ve své relaci.
- Cizí klíč - jednoznačně identifikuje záznam v cizí relaci.
- Stejnou hodnotou v primárním klíči relace A a cizím klíčem v relaci B odkazujícím na relaci A vyjadřuje vztah záznamu z relace B k relaci A.
- Pro každou hodnotu cizího klíče musí existovat ta samá hodnota v odpovídajícím primárním klíči. Cizí klíč se nesmí odkazovat neexistující záznam.

Referenční integrita

- tabulka Zaměstnanci ("osobní číslo" PK, ...
- tabulka Děti(...,"osobní číslo rodiče" FK, ...
- Každé dítě musí mít existujícího rodiče (zaměstnance) a vztah se vyjádří použitím osobního čísla z tabulky Zaměstnanci, nebo rodiče mimo množinu zaměstnanců - a pak se do sloupce "osobní číslo rodiče" zapíše hodnota NULL.

Číselník

- tabulka o 2-n sloupcích
- Slouží k převedení krátkých kódů na lidsky čitelný text.
- Vymezuje množinu platných kódů - co není v číselníku není platnou hodnotou - seznam zemí, seznam krajů, seznam profesí.

Číselník
ukázka - třípísmenné kódy států

kód	název	name
CZE	Česká republika	Czech Republic
DEU	Německo	Germany
DJI	Džibuti	Djibouti
DMA	Dominika	Dominica
DNK	Dánsko	Denmark

Doménová integrita

- Každý sloupec může obsahovat pouze hodnoty, které jsou smysluplné pro odpovídající atribut - tam, kde má být číslo, bude číslo, kde má být datum bude datum atd.
- Rozsah přípustných hodnot lze blíže upřesnit pomocí podmínek CHECK - např. kladná čísla, čísla z intervalu 0..100, datum po roce 1970, .. .
- Hodnoty se kontrolují při vkládání - při dotazování si musíme být jisti jejich kvalitou.

Základní datové typy

- integer - celá čísla
- numeric - číslo uložené s deklarovanou přesností
- varchar - řetězec znaků s variabilním limitovaným počtem znaků
- text - řetězec znaků s neomezeným počtem znaků
- date - datum (ve dnech)
- time - čas (s přesností na mikrosec)
- timestamp - (kombinace date/time)
- serial - samoplnící se unikátní celočíselná hodnota

Zápis hodnot

- Čísla - tak jak je píšeme obyčejně, s desetinou tečkou - 10, 20, 3.14
- Ostatní - zapisuje se do jednoduchých apostrofů - 'Pavel', '2012-12-31'

Základní funkce

- upper('Ahoj') - převod na velká písmena
- lower('Ahoj') - převod na malá písmena
- substring('Ahoj' FROM 1 FOR 2) - podřetězec
- trim(' Ahoj ') - ořeže mezery
- CURRENT_DATE - aktuální den
- CURRENT_DATE + 1 - následující den
- CURRENT_DATE - 1 - předchozí den
- CURRENT_TIMESTAMP - aktuální čas
- CURRENT_TIMESTAMP + interval '2 hours' - +2 hodiny

Základní funkce

- extract(dow FROM CURRENT_DATE) - den v týdnu
- extract(day FROM CURRENT_DATE) - den v měsíci
- extract(month FROM CURRENT_DATE) - vrací měsíc
- extract(year FROM CURRENT_DATE) - vrací rok
- date_trunc('month', CURRENT_DATE) - vrací začátek daného období
- random() - vrátí náhodné desetiné číslo z intervalu (0, 1)
- to_char(now(), 'DD.MM.YYYY') - formátuje datum

Základní operátory

- =, <>, <, >, <=, >=
- +, -, *,
- || - spojení dvou řetězců
- LIKE - nazev LIKE 'Be%' - test na shodu znaků
- ILIKE - nazev ILIKE '%be%' - nebere ohled na velikost písmen
- :: - přetypování
- / - celočíselné dělení, pokud dělenec a dělitel jsou celá čísla
- / - klasické dělení, pokud dělitel nebo dělenec není celé číslo
- IS [NOT] NULL - test na hodnotu NULL

Základní logické operátory

- **AND** - "a zároveň platí"

  jmeno = 'Pavel' **AND** prijmeni = 'Stěhule'

- **OR** - "nebo"

  profese = 'Programátor' **OR** profese = 'Dispečer'

  profese IN ('Programátor', 'Dispečer')

- **NOT** - "negace"

  NOT profese IN ('Programátor', 'Dispečer')

Příkaz SELECT

- Nejdůležitější v SQL.
- Je nutné si zapamatovat posloupnost a význam jednotlivých klauzulí.
- Pokud je výsledkem víceřádková relace, tak by měl obsahovat klauzuli ORDER BY.

Příkaz SELECT
základní syntaxe

```
SELECT co
    FROM odkud
  WHERE co_nas_zajima
  GROUP BY podle_ceho_agregujeme
  HAVING jak_chceme_omezit_agregace
  ORDER BY podle_ceho_radit DESC
  LIMIT kolik_chceme_radku
```

Příkaz SELECT
ukázky

- Výběr nejstarší osoby z tabulky Osoby

```
SELECT *
    FROM osoby
  ORDER BY vek DESC
  LIMIT 1
```

- Výběr všech osob se jménem Pavel

```
SELECT *
    FROM osoby
  WHERE jmeno = 'Pavel'
```

Příkaz SELECT
cvičení I.

- Vyberte nejmladší osobu.
- Vyberte všechny Zdeňky.
- Dohledejte nejmenší obec z tabulky Obce.
- Dohledejte největší obec.
- Vyberte všechny obce s počtem obyvatel do 10 tisíc obyvatel.
- Vyberte všechny obce, kde je víc žen než mužů a mají více než 50 tisíc obyvatel - spojování podmínek logickým operátorem AND.
- Vyberte 10 obcí s největším poměrem počtu žen vůči počtu mužů.

Řešení dvou posledních úkolu

```
SELECT *
   FROM obce
  WHERE pocet_zen > pocet_muzu
    AND pocet_zen + pocet_muzu > 50000;
```

```
SELECT *
   FROM obce
  ORDER BY pocet_zen/(pocet_muzu::float) DESC
  LIMIT 10;
```

Příkaz SELECT
spojování relací

- Z implementačních důvodů, viz důvody pro normální formy, musíme (téměř vždy) získat data z více databázových relací (tabulek).
- Příkaz **JOIN** v klauzuli **FROM** umožňuje párování záznamů z dvou různých relací *A*, *B*.
- Po vytvoření kartézského součinu relací *A* x *B* se ponechají záznamy vyhovující filtrující podmínce uvedené za klíčovým slovem **ON**.
- *A* **JOIN** *B* **ON** A.pk = B.fk
- *Pozor – při nesmyslně navržené filtrující podmínce je výsledkem nesmysl. DB nekontroluje smysluplnost podmínky.*

Příkaz SELECT
spojování relací/ukázky

- Ke každému zaměstnaci zobraz název jeho profese

```
SELECT o.*, p.nazev
    FROM osoby o
        JOIN profese p
        ON p.id = o.profese_id
```

- Vyber všechny programátory

```
SELECT o.*
    FROM osoby o
        JOIN profese p
        ON p.id = o.profese_id
  WHERE p.nazev = 'programator'
```

Příkaz SELECT
spojování relací/ukázky

- Dohledejte faktury podané v lednu včetně osob, které je fakturovaly

```
SELECT f.*, o.jmeno, o.prijmeni
    FROM faktury f
        JOIN osoby
        ON f.podal_osoby_id = o.id
  WHERE f.podano BETWEEN '2012-01-01'
                     AND '2012-01-31'
```

Příkaz SELECT
cvičení 2.

- Z tabulky osoby vyberte všechny dispečery
- Kdo je nejstarší programátor?
- Vyberte všechny obce z okresu Benešov
- Vyberte 10 obcí s nejvíce obyvateli z okresu Benešov
- V které obci je největší podíl žen vůči můžům na okrese Benešov?

Řešení posledního úkolu

```
SELECT ob.*
    FROM obce ob
            JOIN okresy ok
            ON ob.okres_id = ok.id
   WHERE ok.nazev = 'Benešov'
   ORDER BY ob.pocet_zen/(ob.pocet_muzu::float) DESC
   LIMIT 1
```

Příkaz SELECT
agregace - GROUP BY, HAVING

```
SELECT co
    FROM odkud
  WHERE co_nas_zajima
  GROUP BY podle_ceho_agregujeme
  HAVING jak_chceme_omezit_agregace
  ORDER BY podle_ceho_radit DESC
  LIMIT kolik_chceme_radku
```

Příkaz SELECT
agregace

- Relace můžeme rozdělit na podmnožiny na základě stejné hodnoty v jednom nebo více atributu - klauzule GROUP BY.
- Bez použití klauzule GROUP BY je relace jednou podmnožinou obsahující všechny záznamy.
- Na každé podmnožině můžeme spočítat agregační funkci(e) min/max/sum/count/...
- Výsledek těchto funkcí si můžeme zobrazit spolu s hodnoutou určující odpovídající podmnožinu. Po použití agregační funkce se zvyšuje minimální detail na úroveň množin - tj lze zobrazit pouze výsledky agregačních funkcí nebo sloupce, podle kterých se agreguje.

Příkaz SELECT
agregační funkce

- min - minimum z množiny
- max - maximum z množiny
- avg - průměr z množiny
- count - počet prvků v množině
- sum - součet prvků v množině
- string_agg - součet řetězců
- ....

Příkaz SELECT
agragace/ukázky I.

- Kolik osob je evidováno v db?

```
SELECT count(*)
    FROM Osoby
```

- Kolik programátorů je evidováno v db

```
SELECT count(*)
    FROM Osoby o
        JOIN profese p
        ON p.id = o.profese_id
  WHERE p.nazev = 'programator'
```

Příkaz SELECT
agreace/cvičení 3.

- Kolik obcí je v okrese Benešov?
- Zobrazte počet obyvatel podle okresů a seřaďte je podle velikosti od největšího
- Který okres v ČR má nejvíce obyvatel
- Který okres v ČR má nejvíce obcí
- V kterém okrese žije nejvíce obyvatel v obcích nad 10000 obyvatel?
- Zobrazte průměrné velikosti obcí podle okresů a seřaďte je podle velikosti od největšího.

Řešení poslední úlohy

```
SELECT avg(pocet_muzu + pocet_zen)::int,
       ok.nazev
  FROM obce ob
       JOIN okresy ok
       ON ob.okres_id = ok.id
 GROUP BY ok.nazev
 ORDER BY avg(pocet_muzu + pocet_zen) DESC,
          ok.nazev;
```

Příkaz SELECT
agregace/klauzule HAVING

- Klauzule **WHERE** - filtr na surových datech
- Klauzule **HAVING** - filtr na agregovaných datech

- Které okresy mají více než 1 mil obyvatel?

```
SELECT sum(pocet_muzu+pocet_zen), k.nazev
    FROM obce o
            JOIN okresy k
            ON k.id = o.okres_id
  GROUP BY k.nazev
  HAVING sum(pocet_muzu+pocet_zen) > 1000000
```

Příkaz SELECT
agregace/cvičení 4.

- Ve kterých okresech jsou obce nad 0.5M obyvatel?
- Které okresy obsahují méně než 1000 obcí?

Příkaz SELECT
analytické funkce

- Podobně jako u agregačních funkcí se počítají nad podmnožinou
- Oproti agregačním funkcím nedochází k redukci detailu na úroveň podmnožin
- Pro začátek nejdůležitější analytické funkce jsou row_number a rank (případně dense_rank)
- Syntaxe - **fce(..) OVER (PARTITION BY .. ORDER BY .. DESC)**
- Pro analytické funkce neexistuje obdoba klauzule HAVING - pro filtrování je nutné použít tzv *derivovanou tabulku.*

Derivovaná tabulka

- SELECT ze SELECTu
- V klauzuli FROM se objeví příkaz SELECT zapsaný v závorkách - (pozn. vstupem příkazu SELECT jsou relace - ta je i výsledkem příkazu SELECT - tudíž lze napsat SELECT ze SELECTu
- Derivovaná tabulka MUSÍ MÍT alias

  SELECT * FROM (SELECT ...) x

Příkaz SELECT
analytické funkce/ukázky

- Zobrazte přehled, kde jsou zobrazeni všichni zaměstnanci seřazení podle věku dle profese

```
SELECT row_number()
OVER (PARTITION BY p.nazev
        ORDER BY o.vek DESC),
         o.*
FROM osoby o
        JOIN profese p
        ON p.id = o.profese_id
ORDER BY p.nazev, 1
```

Příkaz SELECT
analytické funkce/ukázky

- Zobrazte dva nejstarší zaměstnance z každé profese

```
SELECT x.jmeno, x.prijmeni, x.vek, x.profese
    FROM (SELECT rank()
                    OVER (PARTITION BY p.nazev
                            ORDER BY vek DESC),
                          o.jmeno, o.prijmeni, o.vek,
                          p.nazev AS profese
                FROM osoby o
                        JOIN profese p
                        ON p.id = o.profese_id) x
  WHERE x.rank <= 2
  ORDER BY x.profese, x.vek DESC
```

Příkaz SELECT
analytické funkce/cvičení 5.

- Z každého okresu vyberte seznam 10 obcí s největším počtem obyvatel.

Příkaz SELECT
jednoduché vnořené dotazy

- Příkaz SELECT může obsahovat další (vnořené) příkazy SELECT
- V nejjednodušším případě nahrazují konstantu, kterou můžeme získat jiným příkazem SELECT - konstantu můžeme nahradit přímo příkazem zapsaným do závorek
- Pro jednoduchost lze poddotaz nahradit tabulkou - příkaz CREATE TABLE AS SELECT nebo pohledem - příkaz CREATE VIEW AS SELECT

Příkaz SELECT
jednoduché vnořené dotazy

- Zobrazte osoby jejichž věk je vyšší než průměrný

1) SELECT AVG(vek) FROM osoby => prum_vek

2) SELECT * FROM osoby
   WHERE vek > prum_vek

- Kombinace - poddotaz

```
SELECT *
  FROM osoby
 WHERE vek > ( SELECT AVG(vek)
                 FROM osoby )
```

Příkaz SELECT
jednoduché vnořené dotazy/cvičení

- Získejte seznam okresů s větším počtem obyvatel než je průměr v ČR.

Příkaz SELECT
LEFT/RIGHT JOIN

- Výsledkem spojení relací jsou pouze záznamy u kterých došlo ke zpárování.
- Záznamy u kterých nedošlo ke zpárování "zmizí".
- Tzv vnější spojení relací (outer JOIN) umožňuje zobrazit i tyto nezpárované záznamy a to z levé relace (LEFT JOIN) nebo z pravé relace (RIGHT JOIN) vztaženo ke klíčovému slovu JOIN.
- Často se dohledává pro dohledání nezpárovaných záznamů - chybějící hodnoty se nahradí konstantou NULL.

NULL

- Pozor NULL je "zvláštní".
- Slouží pro vyjádření neexistující, neznámé, nedefinované hodnoty.
- NULL cokoliv je NULL - takže pro informaci, zda je něco NULL nelze použít =
- Nicméně existuje bezpečný operátor IS NULL.
- a existuje bezpečná funkce COALESCE, která vrací první ne NULL hodnotu.
- V databázi se snadno dotazuje na to, co tam je. Naopak, to co tam není, to se dohledává obtížněji.

Příkaz SELECT
LEFT JOIN/ukázka

- S využitím tabulky fakturace dohledejte, kdo nefakturoval v lednu 2012?

```
SELECT o.*
    FROM osoby o
            LEFT JOIN (SELECT *
                              FROM fakturace
                             WHERE f.podano BETWEEN '2012-01-01'
                                                AND '2012-01-31') f
            ON o.id = f.podal_osoby_id
WHERE  f.id IS NULL
```

Příkaz SELECT
NOT IN (vnořený SELECT)

- Předchozí příklad lze napsat jednodušeji pomocí vnořeného SELECTu (subselectu)

```
SELECT *
    FROM osoby
  WHERE id NOT IN (SELECT podal_osoby_id
                           FROM faktury
          WHERE f.podano BETWEEN '2012-01-01'
                                 AND '2012-01-31')
```

- Jelikož vnořený SELECT může vrátit více hodnot je nutné použít operátor testu na neexistenci prvku v množině NOT IN (nikoliv <>)

Příkaz SELECT
rekapitulace

```
SELECT co
    FROM odkud
  WHERE co_nas_zajima
  GROUP BY podle_ceho_agregujeme
  HAVING jak_chceme_omezit_agregace
  ORDER BY podle_ceho_radit DESC
  LIMIT kolik_chceme_radku
```

Příkaz INSERT

- Slouží pro vložení nových záznamů do tabulky
- Můžeme vložit konstantní záznam

```
INSERT INTO tabulka(sloupce)
       VALUES(hodnoty)
```

- Můžeme vložit relaci, která vznikne jako výsledek příkazu SELECT

```
INSERT INTO tabulka(sloupce)
    SELECT ...
```

Příkaz UPDATE

- Slouží k aktualizaci existujících hodnot v tabulce
  - Aktualizace věku osob

```
UPDATE osoby
   SET vek = vek + 1
```

- POZOR - NEEXISTUJE UNDO!

Ukázka špatně navržené databáze - pro správnost údajů je nutné každý den neustále kontrolovat a aktualizovat databázi. Správným řešením je věk dopočítávat a do db uložit datum narození

Zákaznická funkce *pro výpočet stáří z data narození*

```
CREATE OR REPLACE FUNCTION vek (date)
RETURNS int AS $$
SELECT EXTRACT(YEAR FROM age($1))::int
$$ LANGUAGE sql;
```

```
postgres=# SELECT vek('1981-10-10');
 vek
-----
  31
(1 row)
```

Příkaz DELETE

- Odstraní záznamy z tabulky
  - Smažte všechny programátory

```
DELETE FROM osoby
  WHERE profese_id = (SELECT id
                      FROM profese
                      WHERE nazev = 'programator')
```

- Pozor NEEXISTUJE UNDO!

Příkaz SELECT
rekapitulace

```
SELECT co
    FROM odkud
  WHERE co_nas_zajima
  GROUP BY podle_ceho_agregujeme
  HAVING jak_chceme_omezit_agregace
  ORDER BY podle_ceho_radit DESC
  LIMIT kolik_chceme_radku
```

Otázky?